Image Super Resolution using Deep Convolutional Neural Networks

SPECTRUM

Rithvika Pervala, 11940830 Shubham Gupta, 11941140

Abstract

Every single day we come across the problem of unclear and distorted images making it extremely difficult to work in such situations. There is a need for high resolution images for the purpose of analysis, research and development in modern scenarios. There is a need for a clean mapping from low resolution inputs to high-resolution images.

To optimize the processes, we do not use the traditional CNN which associates each feature and enhances it. The Deep CNN restores quality by an efficient trade-off between performance and time within a lightweight structure. We also seek to enhance the colour coding from the traditional RGB space to the more vivid $\mathrm{YC_bC_r}$ colour space.

Introduction

The problem of Super-Resolution (SR) is a persistent one that has created the problem of distorted images for decades. Many researchers over the years have tried to come up with solutions. The solution to the problem of SR is not a single one. Breaking images, enhancing them and then aggregation is one way. Enhancing each functionality as a whole and then overlapping all of them is another way.

We use a more optimized version for SR where the CNN directly maps the low-resolution image to the high-resolution image CNN. We embed each functionality as a dictionary in the hidden layers of the CNN. The patch and image extraction are also formulated by the CNN.

We were given the BSDS500 images dataset in high-resolution format. We applied distortion in these images using Gaussian filter and pre-processed them and classified them as low-resolution images. The initial images become the target images i.e. low-resolution mapped to high-resolution images. This was done both for the training and validation set. The dataset was loaded into the CNN and results were predicted.

Problem Definition

Given a source image with low resolution, we need to give an image that has a high resolution than the input source image.

Objective

- The target is to build a CNN (preferred 3 layered) that incorporates all the functionality from the feature extraction, resolution enhancement, classification to the error analysis.
- The concepts of the **Mean Square Error** (MSE) loss, **Stochastic Gradient Descent** (SGD)model optimizer and **Rectified Linear Unit** (ReLU) activation function were used.

Technology Used

- Google Colab
- NVIDIA GPU CUDA 4.0
- Google Drive
- Github
- Image Reader

Libraries Used

- Standard Python Libraries for I/O and System Operations
- python=3.6.8
- h5py==2.10.0
- ipykernel==5.1.3
- Markdown==3.1.1
- matplotlib==3.1.2
- numpy==1.17.4
- scikit-image==0.16.2
- scipy==1.3.3

- ipython==7.9.0
- ipython-genutils==0.2.0
- jupyter-client==5.3.4
- jupyter-core==4.6.1
- tensorboard==2.0.1
- torch==1.3.1
- torchvision==0.4.2
- tqdm==4.40.0

Problems Faced

Code and Documentation Issues

- The **GitHub Repository** that we used for reference is badly documented and hence we faced issues in understanding the code
- We later documented the code and cleaned it on the repo to make it readable and clear.
- Wrote comments for all the source codes, user-defined functions (`distort_image()`, `evaluate_model()`, `ycbcr2rgb()` , `psnr()`) and data pre-processing.

Unsupported Image Formats

- The initial code didn't support PNG format in `evaluate_model()` function and since we used images of that format for **Evaluation** ($\mathrm{Set5}$, $\mathrm{Set14}$ and $\mathrm{BSDS200}$) **Dataset**
- We added the support later on along with the pre-existing JPEG and BMP after we realized the issue.

Computation Issues

- The Model took a lot of time to train as we run over the whole data in multiple epochs.
- Training for 20 epochs needed approximately 7 hours despite using **Google Colab** with dedicated GPU for the computation.
- Colab also has an issue of disconnecting in case of prolonged periods of computation with GPU for server conservation.
    - Hence this lead to an issue where we have got disconnected in the middle of the Model Training a couple of times.

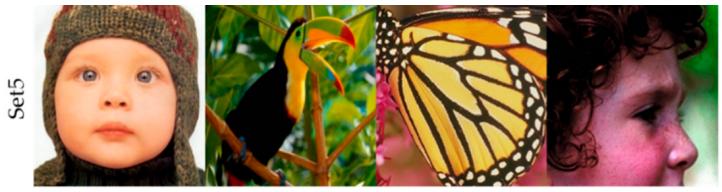
Dataset Used

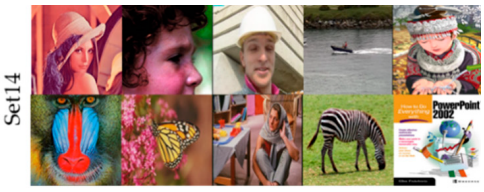
Berkeley Segmentation Dataset 500 (BSDS500)

- The dataset consists of 500 natural images, ground-truth human annotations and benchmarking code. The data is explicitly separated into disjoint train, validation and test subsets.
- The dataset is an extension of the BSDS300, where the original 300 images are used for training / validation and 200 fresh images, together with human annotations, are added for testing. Each image was segmented by five different subjects on average.

Set5, Set14

- Set5 (Consisting of 5 images) and Set14 (Consisting of 14 images) is a standard dataset to evaluate the performance of super resolution of images.





BSDS200

- BSDS200 is an older version of BSDS 500 comprising of less and old images.

- **Training Dataset** : $\mathrm{BSDS500}$ Training and Testing Set combined
  - This is for training the machine learning model.
- **Validation Dataset**: $\mathrm{BSDS500}$ Validation Set
  - Validation determines the accuracy based on the machine learning model training
- **Evaluation Dataset**: Combined dataset of $\mathrm{Set5}$, $\mathrm{Set14}$ and $\mathrm{BSDS200}$.
  - This is mainly used for the comprehensive error analysis and is unrelated to the initial dataset. It is used for the prediction of real-world data. It's main purpose is to give a comprehensive analysis about situations when the model will be used on real-world data which will generally be different in formats, resolution and quality.

Models Used

- Our SRCNN Pytorch Model consists of 3 CNN Layers that are used for processing the Images with the given parameters
- These three layers are normalized using standard mean and deviation and the bias are initialized to 0.
  - Mean - 0
  - Standard Deviation - 0.001
  - Biases - 0

```
<bound method Module.named_parameters of SRCNN(
  (conv1): Conv2d(1, 64, kernel_size=(9, 9), stride=(1, 1), padding=(4, 4))
  (conv2): Conv2d(64, 32, kernel_size=(1, 1), stride=(1, 1))
  (conv3): Conv2d(32, 1, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3))
)>
```

SRCNN Pytorch Model Parameters

1^{st} CNN Layer - Patch Extraction and Representation

- **Specifications**
  - This layer has Kernel Size dimensions 9×9
  - In Channels - 1

- Out Channels - 64
- Padding - 4

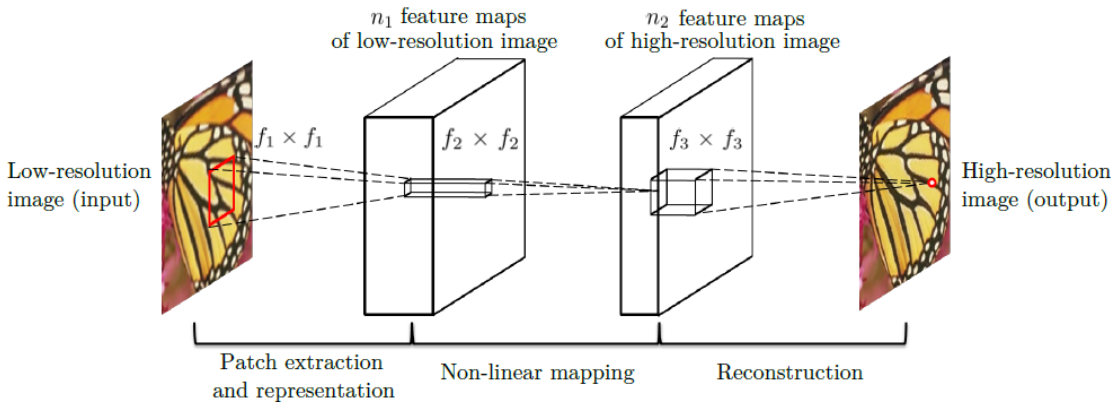
- This layer **densely extracts overlapping patches** from the **Interpolated Images** and **represents** each patch as a **High-(n_1)dimensional Vector**
- **ReLU** Activation Function is applied in this layer as otherwise it would become purely **Linear Convolution**

2^{nd} CNN Layer - Non-Linear Mapping

- **Specifications**
  - This layer has Kernel Size dimensions 1×1
  - In Channels - 64
  - Out Channels - 32
  - Padding - 0
- This **Hidden Layer Nonlinearly maps** each **Interpolated high-(n_1) dimensional vector** onto another **High Resolution high-(n_2)dimensional vector**. These vectors comprise another set of feature maps.
- **ReLU** Activation Function is applied in this layer as well.

3^{rd} CNN Layer - Reconstruction

- **Specifications**
  - This layer has Kernel Size dimensions 7×7
  - In Channels - 32
  - Out Channels - 1
  - Padding - 3
- This layer is used aggregate all the mapped **High Resolution high-dimensional vectors** to generate the **high resolution image output $F(Y)$.**



Implementation

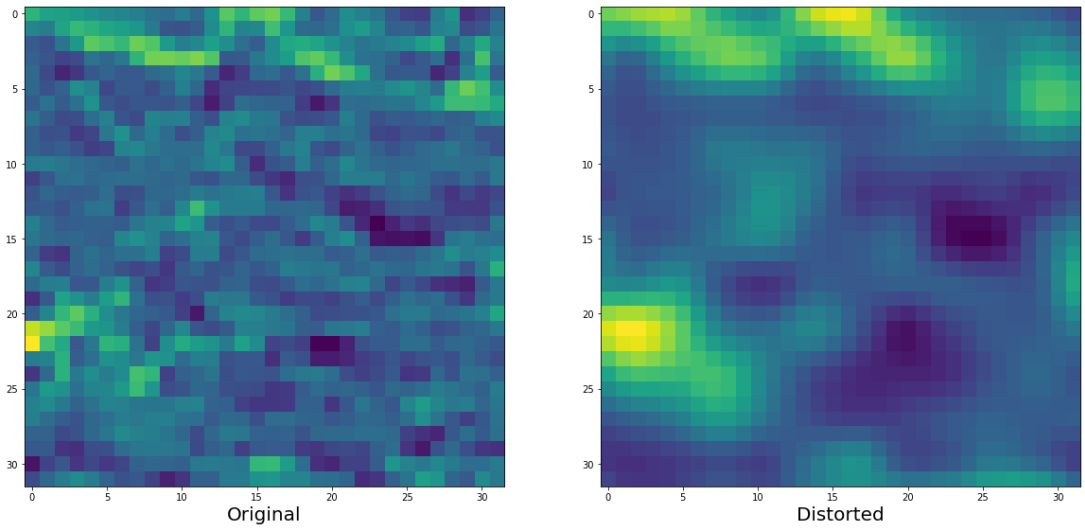
Overview

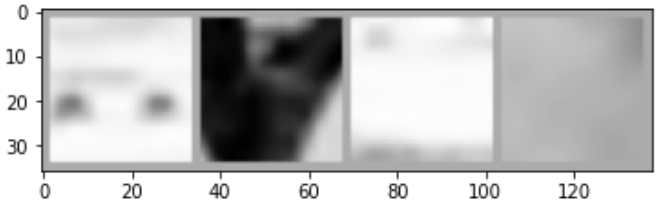
- We **distort** the **Ground Truth High Resolution** Image X to a Interpolated Image Y and send it to our Trained SRCNN Model to reconstruct.
- The Trained Model creates an image $F(Y)$ - Goal is its should be as similar as possible to **Ground Truth.**
- Model needs to learn the mapping F that leads to that - this is done via several CNN layers (in our case 3).

Preprocessing

- $\mathrm{YC_bC_r}$ **Colour Spacing** - We start out by going through the **Ground Truth Train, Test, Validation Sets** of $\mathrm{BSDS500}$ to generate the target images in a tensor by walking through the directories, converge each image into a Square `numpy` Matrix and storing them in $\mathrm{YC_bC_r}$ colour spacing.
- **Distortion and Upscaling/Interpolation** `distort_image()`**.**
  - **Distortion -** We distort **Train, Test and Validation Sets** Images by first separating the colour channels and applying distortion in Y colour channel with the help of the custom function `distort_image()` using a blurring factor SIGMA for **Gaussian Blurring**.
  - **Upscaling -** The **Low-resolution distorted** Image is then upscaled to same size as **Ground Truth** using **bicubic interpolation** and **Super Resolution Factor** and we return this upscaled image from `distort_image()`**.**

- **Verification -** We then verify whether the **Distortion** and **Interpolation** is done properly by randomly Plotting 10 **training images.**





4 Interpolated Training Images in Grey Scale

- **Normalization -** We then Normalize our **Train, Test, Validation Sets** with **mean** and **deviation** of the **Training Images.**

```
0.43398995682829317
0.23199219136380714
```

Standard Mean and Deviation

- **Data Loading -** We build a Tensor Dataset out of the verified **Train, Test, Validation Sets** Target Images so that we can use them later in the SRCNN Pytorch Model.

Loss Function - MSE

- We use MSE **(Mean Square Error)** as a loss function - **Stochastic Gradient descent (SGD)** with **Back Propagation**

- Only calculated between the **central pixels** of the Ground Truth and the Model output

Learning Rate

- First two layers - 10^{-4}
- Last Layer - 10^{-5} - Smaller rate is important for network to converge (**Empirical Observation)**

```
optimizer = optim.SGD(
                    [
                        {"params": SR_model.conv1.parameters(), "lr": 0.0001},
                        {"params": SR_model.conv2.parameters(), "lr": 0.0001},
                        {"params": SR_model.conv3.parameters(), "lr": 0.00001},
                    ], momentum=0.9)
```

GSD Initialization and Learning Rates for each CNN Layer

Evaluation Metrics

PSNR (Peak Signal Noise Ratio)

- PSNR is a widely used metric for quantitatively evaluating high **restoration quality**
- Partially related to **perceptual quality** as well
- PSNR is usually expressed as a logarithmic quantity using the **decibel scale (dB)**.
- MSE is known to favour High PSNR value

SSIM (Structural Similarity Index Measure)

- SSIM is used to quantify **Perceptual Quality -** How humans perceive it.
- It ranges between $[0, 1]$
- Although this metric better evaluated over MAE **(Mean Absolute Error),** MSE also gives satisfactory performance over SSIM**.**

Model Fitting, Optimization, Evaluation

Training

- We run through the **Interpolated Training Set** multiple times for 20 Epochs and train the above SRCNN estimator to give us a Trained Model
- Inside each epoch we find the MSE loss, **back propagate** for optimization and carry forward step by step using the **Stochastic Gradient descent** optimizer

Validation

- Then we traverse through **Interpolated Validation Set,** find the minimum **validation** MSE loss by constantly storing the lowest loss found as of yet into a variable.
- As soon as a lower loss is found, it makes a check point and stores the information.

Metric Evaluation

- We then evaluate then the **mean PSNR** and **SSIM** values per epoch for **Bicubic Interpolated Images** and **Super Resolution Output Images** on the Y colour channel using a custom function `evaluate_model()`
- We do the evaluation over $\mathrm{Set5}$, $\mathrm{Set14}$ and $\mathrm{BSDS200}$
- In the below two images we can see how the Loss, Evaluation Metric values change from epoch 2 to 20

```
epoch:  2
[2,   100] loss: 0.998
[2,   200] loss: 1.003
[2,   300] loss: 0.999
[2,   400] loss: 0.995
[2,   500] loss: 0.992
[2,   600] loss: 1.001
[2,   700] loss: 1.011
[2,   800] loss: 1.007
[2,   900] loss: 0.998
[2,  1000] loss: 1.000
[2,  1100] loss: 1.010
[2,  1200] loss: 0.999
[2,  1300] loss: 1.002
[2,  1400] loss: 0.995
[2,  1500] loss: 0.997
[2,  1600] loss: 0.990
epoch 2 validation loss: 1.036146
PSNR Evaluation over Set5, Set14 and BSDS200 : Bicubic = 25.740662 ; SRCNN = 13.330092
SSIM Evaluation over Set5, Set14 and BSDS200 : Bicubic = 0.729258 ; SRCNN = 0.360187
```

Loss and Evaluation Metric values after Epoch 2

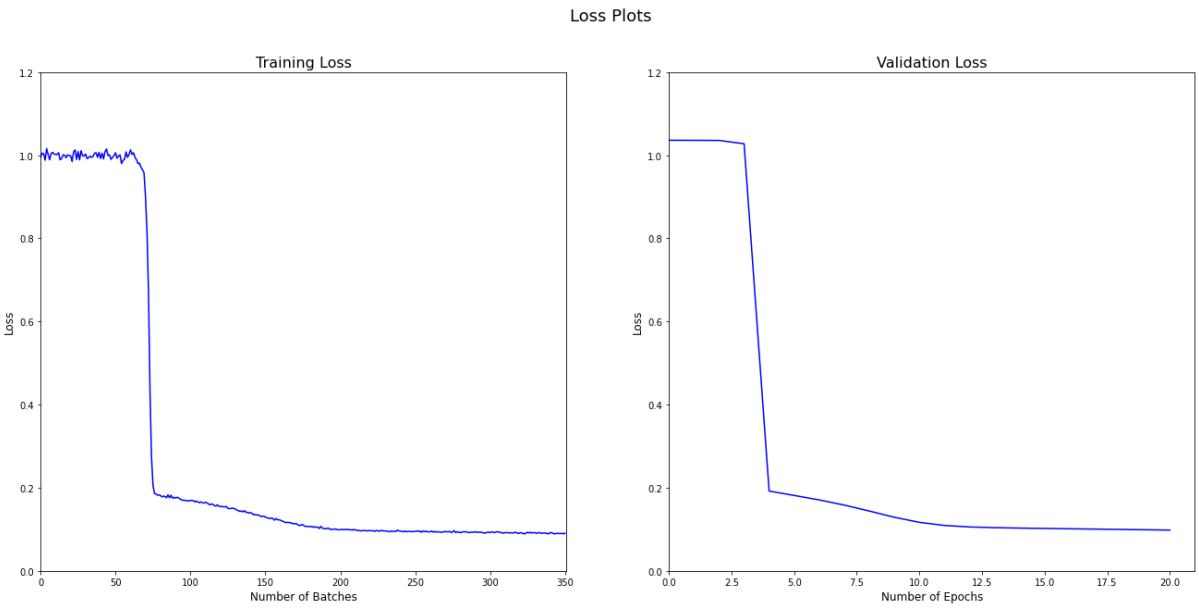
```
epoch:  20
[20,   100] loss: 0.091
[20,   200] loss: 0.092
[20,   300] loss: 0.092
[20,   400] loss: 0.093
[20,   500] loss: 0.093
[20,   600] loss: 0.094
[20,   700] loss: 0.093
[20,   800] loss: 0.093
[20,   900] loss: 0.092
[20,  1000] loss: 0.093
[20,  1100] loss: 0.093
[20,  1200] loss: 0.092
[20,  1300] loss: 0.091
[20,  1400] loss: 0.091
[20,  1500] loss: 0.092
[20,  1600] loss: 0.091
epoch 20 validation loss: 0.099288
New lowest loss found. Saving checkpoint.
PSNR Evaluation over Set5, Set14 and BSDS200 : Bicubic = 25.740662 ; SRCNN = 25.136975
SSIM Evaluation over Set5, Set14 and BSDS200 : Bicubic = 0.729258 ; SRCNN = 0.706434
Finished Training
```

Loss and Evaluation Metric values after Epoch 20

Result and Performance

Accuracy - Loss

- Given the resources and time at our hand, our model obtained a minimal **validation loss** of 0.099288. This can be optimized if run for a longer time.
- **Training loss -** The loss across batches drops sharply after 75 batches that is after around $4 - 6$ epochs
- **Validation Loss -** Similar to training loss we see an abrupt drop in loss after $4 - 6$ epochs, showing that the Model is picking up the intricacies in the form of better representations



Evaluation Metric Comparisons

- After running for 20 epochs we get the following observations between upscaling by Bicubic Interpolation and Super Resolution

Metric	Bicubic Interpolation	SRCNN
PSNR (dB)	25.740662	25.136975
SSIM	0.729258	0.706434

- While the SRCNN still leads to lower PSNR and SSIM values for 20 epochs when compared to Bicubic, it will however surpass it after few more epochs as evident in the research paper and the below graph as well.

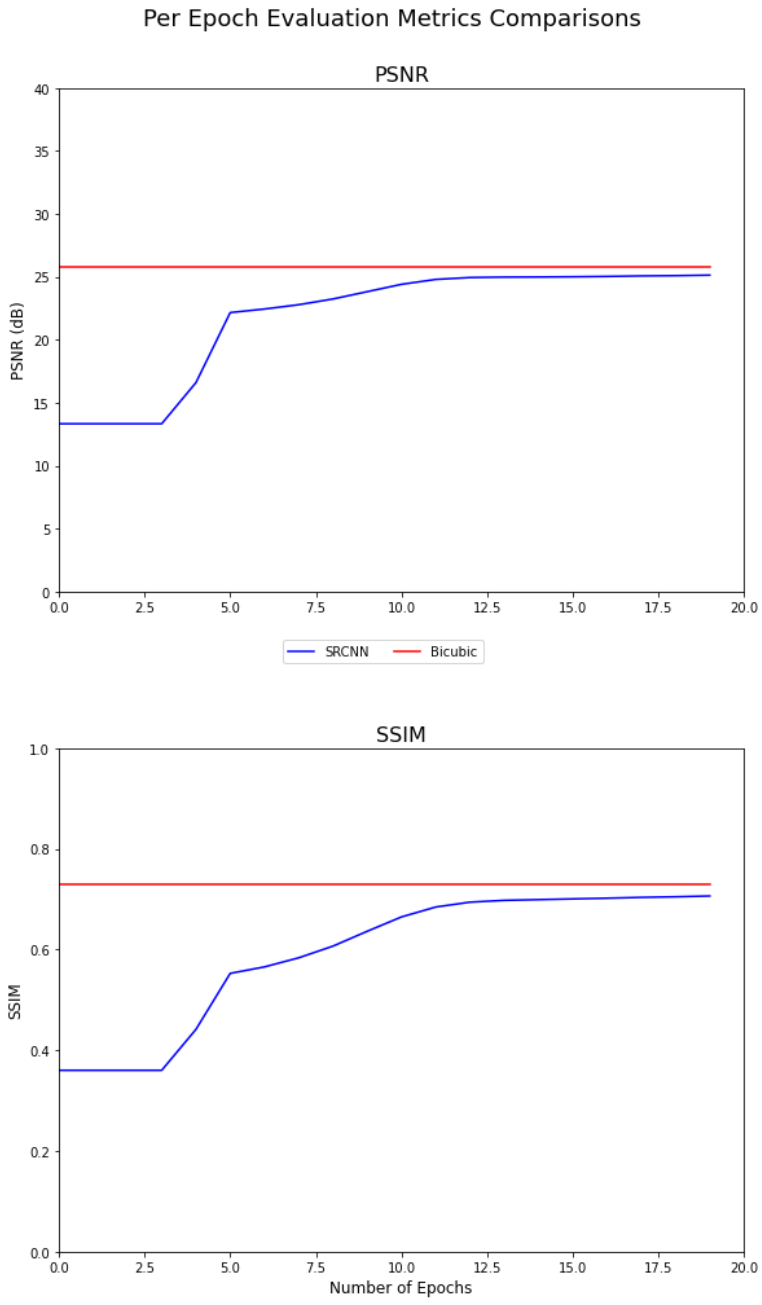
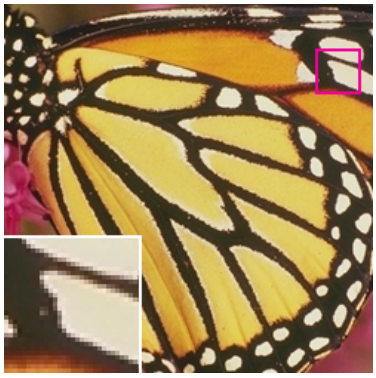
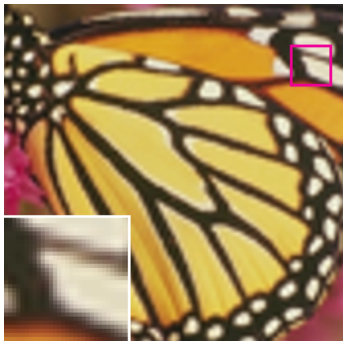


Image Comparison

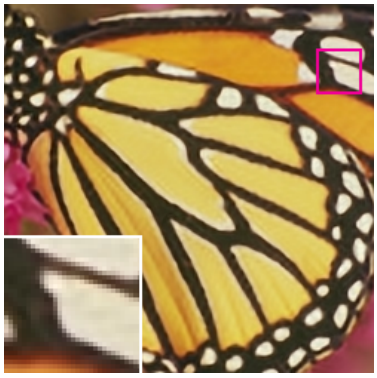
- This is a comparison over an image taken from $\mathrm{BSDS500}$ dataset between bicubic interpolation and SRCNN.



Original



Bicubic Interpolated



SRCNN

Conclusion and Further Work

This SRCNN showed 3 basic steps

- We presented a fully convolutional neural network for image super-resolution. The network directly learns an end-to-end mapping between low and high-resolution images, with little pre/postprocessing beyond the optimization.
- We established a relationship between our deep learning-based SR method and the traditional sparse-coding-based SR methods. This relationship provides a guidance for the design of the network structure.
- We demonstrated that deep learning is useful in the classical computer vision problem of super resolution, and can achieve good quality and speed.
- Overall, we can conclude that the following enhancements gave a optimal performance:
    - There were a host of specification in the **paper** which was not implemented in **GitHub repository** . We have implemented many of them for a more optimum performance and enhanced results.
    - For the evaluation purposes that is the validation, we have added Set5, Set14 and BSDS200 images (a total of 220 images) that gives a combined accuracy of all the datasets. This makes a significant impact on the results. It covers a larger set of distinct images and give results different from the paper and more inclined towards practical applications.
    - The standard RGB color coding was replaced with an enhanced $\mathrm{YC_bC_r}$ color coding which is the main image pipeline in video and digital photography systems. Computers use this color space as it gives a more

vivid and contrasted-color image which is far more pleasing than the RGB image.

- All the pre-processing, noise and error computation and user-defined functions use $\mathrm{YC_bC_r}$ color space (this is also covered in the documentation)
- The paper explicitly specifies that the number of backward propagation is 8×10^8 but the repository implements 15×10^6. Increasing back propagations is not an performance optimization but trains the SRCNN model to capture the features better and increase the resolution in lower epochs.

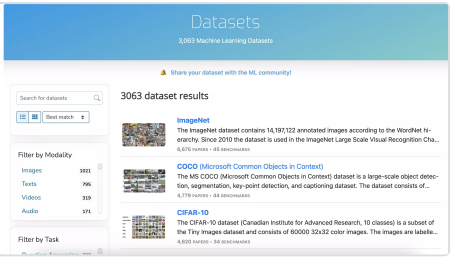
Further Work

- Modularize the code as the code has a lot of blocks that are repeated, especially in the Pre-processing part.
- To complete a comprehensive Error Analysis and graphs whose brief description in the paper but not implemented in the **GitHub repository**.
  - Currently we added the support for SSIM along with pre-existing PSNR
  - We seek to add further functionality for IFC, NQM and other possible evaluation metrics.
  - This includes graph plotting and tabulated results.
- Run the code further for more epochs until SRCNN crosses the **Bicubic Interpolation Benchmark** and to produce quality images that are clearer than before.
- Given more resources , we will seek to enhance the $\mathrm{2D\ CNN}$ to support 128×64 network width from 64×32 width
  - This will enhance the $\mathrm{2D\ CNN}$ to give superior performance even for lower amount of epochs.
- If possible, we might also extend the data used for training, testing and validation by adding support for SR based datasets like $\mathrm{DIV2K}$ (Diverse $\mathrm{2K}$), $\mathrm{Urban100}$, PIRM (Perceptual Image Restoration and Manipulation).

Papers with Code - Machine Learning Datasets

28 datasets * 57577 papers with code.

https://paperswithcode.com/datasets?task=image-super-resolution



References

Papers

- https://arxiv.org/pdf/1501.00092.pdf
- https://www.mdpi.com/2076-3417/11/3/1092/pdf
- https://paperswithcode.com/paper/image-super-resolution-using-deep

Dataset

- https://github.com/Lornatang/SRCNN-PyTorch
- https://www.kaggle.com/ll01dm/set-5-14-super-resolution-dataset
- https://paperswithcode.com/datasets?task=image-super-resolution

Referenced Codes

- https://github.com/abdulwaheedsoudagar/SR-CNN
- https://github.com/tegg89/SRCNN-Tensorflow
- https://github.com/nagadomi/waifu2x
- https://github.com/Lornatang/SRCNN-PyTorch
- https://github.com/highgroundmaster/Image-Super-Resolution-Using-Deep-Convolutional-Networks
- https://github.com/sdv4/SRCNN
- https://github.com/princepansari/Video-SuperResolution_And_GAN_FakeImageDetection
- https://github.com/sdv4/FDPL
- https://keras.io/examples/vision/super_resolution_sub_pixel/